

# Pretty Fast Analysis: An embarrassingly parallel algorithm for biological simulation analysis

David N. LeBard <sup>a,1</sup>,

<sup>a</sup>*Center for Biological Physics, Arizona State University, PO Box 871604, Tempe, AZ 85287-1604*

---

## Abstract

A parallel code has been written in FORTRAN90, C, and MPI for the analysis of biological simulation data. Using a master/slave algorithm, the software operates on AMBER generated trajectory data using either UNIX or MPI file IO, and it supports up to 15 simultaneous function calls. This software has been performance tested on the Ranger Supercomputer on trajectory data of an aqueous bacterial reaction center micelle. Although the parallel reading is poor, the analysis algorithm itself shows embarrassingly parallel speedup up to 1024 compute nodes. At this CPU count the overall scaling of the software compares well NAMD's best reported speedup, and outperforms AMBER's best known scaling by a factor of 3, while using only a small number of function calls and a short trajectory length.

PACS: 82.20.Wt; 87.19.ly; 87.19.rm; 82.39.Jn

*Key words:* Parallel Analysis; Biological Simulation; Embarrassingly Parallel; Biomolecular Dynamics; Molecular Dynamics Analysis

---

## PROGRAM SUMMARY

*Program Title:* Pretty Fast Analysis

*Journal Reference:*

*Catalogue identifier:*

*Licensing provisions:* Standard CPC license, <http://cpc.cs.qub.ac.uk/license/license.html>

*Programming language:* FORTRAN90/C/MPI

*Computer:* Platform independent

*Operating system:* OS independent

*RAM:* Typically a maximum of 250 MB on the nodes and 2 GB on the master

*Number of processors used:* 1 to 1024.

---

<sup>1</sup> Corresponding author

*Supplementary material:*

*Keywords:* Parallel Analysis; Biological Simulation; Embarrassingly Parallel; Biomolecular Dynamics; Molecular Dynamics Analysis

*PACS:* 82.20.Wt, 87.19.ly, 87.19.rm, 82.39.Jn

*Classification:* 3

*Nature of problem:* Currently, several parallel biomolecular simulation packages are capable of generating literally terabytes of trajectory data. However, there has not been a corresponding push to create parallel biomolecular simulation analyzer that can handle large trajectory sets in a reasonable time.

*Solution method:* This software calculates both structural and energetic properties of biomolecules in parallel using a simple master/slave methodology. The program itself is keyword driven, allowing up to 15 different structural and energetic calculations to be performed at once. Trajectories can be read in either binary or ASCII formats, using either typical Unix or MPI IO. This software has been through several revisions, and in its current state it operates well in either a small linux cluster or supercomputer environment. In the past, the code has been used to calculate the thermodynamic and kinetic properties of aqueous tryptophan amino acid [1], plastocyanin [1,2] and the bacterial reaction center [3], where the latter two are canonical proteins studied within the umbrella of protein electron transfer.

*Restrictions:* In the current form, the code only analyzes AMBER trajectory data from the binpos, mdcrd, or modified binpos filetypes. Also, the code has only been tested with Portland Group compilers and therefore mixing other FORTRAN/C compilers is not recommended.

*Additional comments:* The AMBER topology file reader subroutine (readprm), was taken from readprm.f from the AMBER8 distribution and has been used with permission from David Case and the AMBER development team.

*Running time:* Depends on the size of the biomolecule, the number of the surrounding solvent molecules, the nature of the calculations, as well as the number of frames in the simulation trajectory. Generally, a trajectory of 500,000 frames can be analyzed for tens of energetic and structural properties in a matter of hours.

*References:*

- [1] D. N. LeBard, D. V. Matyushov, J. Phys. Chem. B. 112 (2008) 5218
- [2] D. N. LeBard, D. V. Matyushov, J. Chem. Phys. 128 (2008) 155106
- [3] D. N. LeBard, V. Kapko, D. V. Matyushov, J. Phys. Chem. B. (2008) ASAP

# LONG WRITE-UP

## 1 Introduction

For more than 30 years, advancements in biological molecular dynamics (MD) simulations have pushed the limits of computation in terms of parallel scalability [1,2,3] and algorithm optimization[1,4]. Biological systems under study will generally consist of  $N_{sys} \sim 10^4 - 10^6$  atoms when run with medium to large biomolecular polymers in explicit or implicit model solvents. In recent years, data have shown that in order to capture the long time dynamical motions of biological systems [5,6], production runs on the order of nanoseconds to milliseconds are required, stretching the MD performance requirements even further. The simulation algorithms themselves, though very efficient, are highly specialized to achieve maximum performance on the largest number of processors[1] to generate a trajectory of configuration data by numerically solving equations of motion. The basic strategy of every MD software is to parallelize the force calculations by using a spacial or particle-based decomposition, and in some cases using dynamic load balancing to achieve highest level of scalability [1] and consequently, the quickest trajectory generation.

While these advancements have allowed studies of long time dynamics of biomolecular systems with atomistic detail, similar attention has not been paid to the analysis of such large data sets. Trajectories produced from biological simulations contain configuration data of all  $N_{sys}$  atoms in the system for a given number of steps,  $N_{steps}$ , around  $10^5 - 10^6$  for runs in the nanosecond range for a moderate saving frequency. To analyze such trajectories, pairwise interactions between subsets ( $N_{sys} = N_1 + N_2 + \dots$ ) of these atoms are calculated at each timestep such that the total number of calculations, and therefore the analysis time, are on the order of

$$\tau_{ana} \propto O(\gamma N_1 N_2 N_{steps}) \quad (1)$$

Here,  $\tau_{ana}$  is the total analysis time and  $\gamma$  is the number of function calls required for the analysis. In principle,  $N_1$  and  $N_2$  differ for each function call during the analysis procedure. If the number of needed pairwise calculations are large enough, parallelizing the analysis at the  $N_{steps}$  or  $N_{sys}$  level would certainly lead to an strongly scaling algorithm where the analysis time would be reduced to

$$\tau_{ana}(N_{CPUs}) = \tau_{ana}/N_{CPUs} \quad (2)$$

,

where  $N_{CPUs}$  refers to the number of CPUs available for the calculation.

To handle such large number of calculations, a parallel data analysis algorithm has been developed to perform a variety of structural and energetic calculations to decrease the time to solution of various biophysical problems proportional to the number of compute nodes invested. This algorithm, built into the Pretty Fast Analysis (PFA) software, assumes that the total analysis time follows the simple relation:

$$\tau_{tot}(N_{CPUs}) = \tau_{ana}(N_{CPUs}) + \tau_{read} + \tau_{comm} \quad (3)$$

In Eq. 3,  $\tau_{tot}$  is the total software time,  $\tau_{read}$  is the time required for trajectory reading, and  $\tau_{comm}$  is the time for any associated time for communication between processors. One would expect that  $\tau_{tot} > \tau_{ana} > \tau_{comm}$ . By parallelizing trajectory reading analogous to the data analysis (Eq. 2), and by minimizing communication costs such that  $\tau_{comm} \ll \tau_{read}$ , one can see that the speedup (the ratio  $\tau_{tot}(1CPU)/\tau_{tot}(N_{CPUs})$ ) will be equal to  $N_{CPUs}$ . The goal of this paper will be to demonstrate this relationship holds experimentally true using up to thousands of compute nodes for a minimum number of function calls and trajectory steps.

After a brief theoretical overview of the PFA analysis applied to electron transfer theory in Section 2, the software itself will be overviewed in section 3. The software's parallel performance will be shown in section 4, and we will reveal the final conclusions in section 5. For more detailed instructions on running the software, the reader is referred to the PFA website.

## 2 Theoretical Background

This code was born out of necessity to analyze large amounts of simulation data on several photosynthesis proteins. Therefore the type of calculations given here is merely an example from electron transfer theory to provide a real-life example of the extent to which the analysis may be performed in parallel. This section is not meant to be an introduction to electron transfer theory, for that the reader should see the review in Ref. [7], but rather as an introduction to the calculations used as the test of the PFA algorithm.

In electron transfer theory, the first and second cumulants of the vertical energy gap,  $\Delta E$ , are needed to calculate the kinetic and thermodynamic parameters activating the electron transfer process [5,8,6]. The energy gap itself is defined as:

$$\langle \Delta E \rangle = \Delta E^C + \Delta E^I + \Delta E^{gas} \quad (4)$$

The vertical energy gap in Eq. 4 describes the change in electrostatic energy at the enzymatic site during the charge transfer process and has three components, two of which,  $\Delta E^C$  and  $\Delta E^I$ , can be readily calculated from the simulation data. The first term in Eq. 4 represents the Coulomb component of the vertical energy gap, and can be written as

$$\Delta E^C = \sum_j \Delta q_j \phi_j. \quad (5)$$

In this equation, the index  $j$  runs over all atoms from the redox site that can change electronic state, and  $\phi$  represents the potential of the surrounding biological solvent. Free energy surfaces of oxidation-reduction processes can also be quite affected by the high polarizability of certain cofactors found in electron transfer proteins[7,6,9]. In fact, this term can be calculated from a standard force field simulation [6] using an energetic term accounting for the induction forces of the shifting charge. Since atomic polarizabilities of the reaction center can be tabulated, one needs to calculate the induction term of the vertical energy gap,  $\Delta E^I$ , given in the following form:

$$\Delta E^I = - \left\langle \sum_j (\alpha_j/2) [E_{02}^2(r_j) - E_{01}^2(r_j)] \right\rangle. \quad (6)$$

In Eq. 6 the sum runs over all atoms in the redox site, and  $E_{0X}$  refers to the electric field of enzymatic center the protein in the  $X$  state. The electric field is evaluated at a distance  $r_j$  from the active site in the solvent, and  $\alpha_j$  refers to the atomic polarizabilities.

It has recently been suggested that the dynamics of electron transfer follows the fluctuations of the donor-acceptor distances between electron transferring cofactors [10]. To test this theory in simulations, the separation distance between acceptor and donor cofactors,  $\vec{r}_{DA}$ , is required. This is simply a vector subtraction in the center of mass frame is given by

$$\vec{r}_{DA} = \vec{R}_{Donor} - \vec{R}_{Acceptor} \quad (7)$$

Here,  $R_{Donor}$  and  $R_{Acceptor}$  are the mass weighted vectors of the donor and acceptor cofactors, respectively. The same donor-acceptor atoms vary their charges during the redox process, and therefore it is quite useful to monitor the charge transfer dipole moment

$$\Delta \vec{m}_{DA} = \sum_{j \in DA} \Delta q_j \vec{r}_j \quad (8)$$

In Eq. 8,  $\Delta q_j$  are the difference charges and the sum runs over all atoms

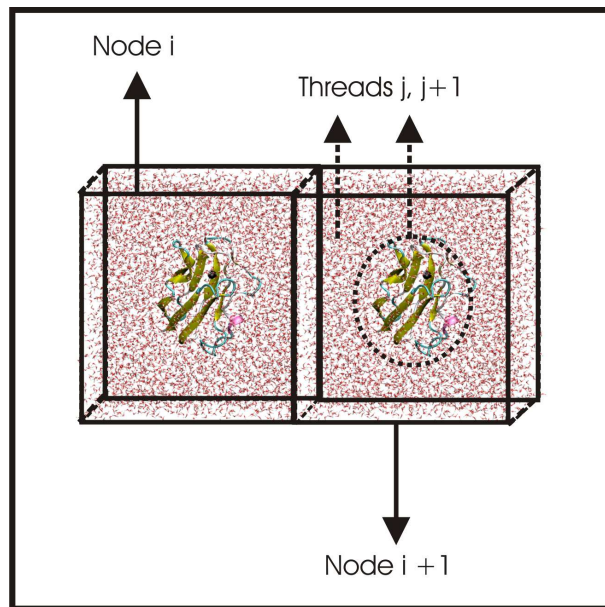


Fig. 1. Schematic of analysis data parallelization. The current version of the algorithm scales strongly and is illustrated with the thick arrows. This method distributes configuration data across the nodes, and each node calculates the all energetics and structural properties of the system. Shown with the dashed arrows, one can see how the data can be further parallelized using threads. In this protocol, properly initialized shared memory threads can calculate pairwise calculations without any send cost, and therefore reduce the analysis time to  $\tau_{ana}(N_{CPUs})/N_{Threads}$ , while concurrently reducing the number of communications by  $1/N_{Threads}$  by splitting the computations into those of the solvent (thread  $j$ ) and the protein (thread  $j+1$ )

in either an electron donor or acceptor cofactor. These equations provide a simple, yet practical example of the efficiency of a highly parallelizable analysis algorithm.

### 3 Overview of the software

The data analysis software has proven to be extremely fast, even over the course of its two year development lifetime due to the simple data splitting procedure (Figure 1). As this figure suggests, the data is distributed such that each node has the entire configuration data for an individual frame of the trajectory. This scheme ensures a minimum communication overhead, which would be incurred if each CPU had less than  $N_{sys}$ , at the cost of storing a dataset on the order of  $O(N_{sys})$ .

At the time of writing, the code consists of 9,950 lines spread over 9 FORTRAN files, and a single file of C code used for .gz file reading. After compiling the

```

If (node = master) then
  Get input from user
  Open all output files
end if

MPI_BROADCAST input to nodes

for (i=1;i=Nsteps-NCPUs+1;i+=NCPUs) do

  If (use MPI IO) then
    MPI_FILE_READ_AT a single frame
  else
    Master reads NCPUs frames into data
    MPI_SCATTER data to nodes
  end if

  Do structural calculations
  MPI_GATHER output on master for writing

  Do energetic calculations
  MPI_GATHER output on master for writing

  If (write output trajectory) then
    Do any coordinate transforms
    MPI_FILE_WRITE_AT a single frame
  end if

end for

If (node = master) then
  Close all output files
end if

```

Fig. 2. Pseudocode of the PFA algorithm. To invoke threads, a decision to use them is taken at input and threads are initialized before the main loop. The threads themselves will need to decide a master thread for each node for sending and receiving the analysis data.

source using the make command, an executable named pfa.e is generated and can be controlled at runtime using a simple keyword driven interface. The code itself is run like,

```
mpirun -np <number of cpus> pfa.e < pfa.in >& pfa.out
```

In its current form, the software can perform up to 15 analysis functions at once including vertical energy gap calculations, vector monitoring, dipole moment calculations, water-shell calculations, mean-squared displacement calculations, as well as AMBER trajectory conversion using MPI IO. The code is capable of reading trajectory data from AMBER's binpos, modified binpos,

Table 1

Timing data for the PFA test on the Ranger supercomputer. All timing data is given in seconds, and has been measured using the WTIME function from the MPI library.

$N_{CPUs}$	$\tau_{read}$	$\tau_{ana}$	$\tau_{tot}$
1	236.6	148909.1	149145.7
16	246.7	9371.0	9617.7
32	92.4	4683.4	4775.8
64	66.3	2349.8	2416.1
128	61.2	1181.9	1243.1
256	67.1	590.2	657.3
512	72.4	294.3	366.7
1024	74.5	147.0	221.4

ASCII mdcrd, or gzipped mdcrd formats with either standard UNIX or MPI IO. It should be noted that Gustafson’s Law [11] allows PFA to scale most strongly when all calculations are made in a single run, which could be a considerable efficiency burst for extremely large systems or those with many atoms in subsets  $N_1$  and  $N_2$ . In the example shown below, the code can be driven to embarrassingly parallel speedup by using even a modest number of function calls (4) for a bacterial reaction center micelle in an explicit TIP3P solvent [12]. For the details of the simulation protocol, the reader is referred to previous work on this system[6].

#### 4 Performance results

The PFA software has been well tested on several clusters ranging in speed from a local Opteron cluster with a gigabit ethernet network, to the Saguaro Supercomputer at Arizona State University running and infiniband network, to the the newly constructed Ranger Supercomputer at TACC. Although this paper focuses on the timing analysis using MPI-IO on the Ranger Supercomputer, similar results have been seen in all three test environments, using UNIX and MPI IO, and the details of such comparisons can be found in the code’s manual.

To test the scalability of the software, a short 12,288 step (245.76 ps) trajectory from a recent simulation of aqueous micelluar bacterial reaction center (BRC) protein[6] was used. The very short trajectory length was determined to be the maximum trajectory length a single CPU could run the 4 functions in a 48 hour period, and therefore was the longest trajectory that could be used. This test

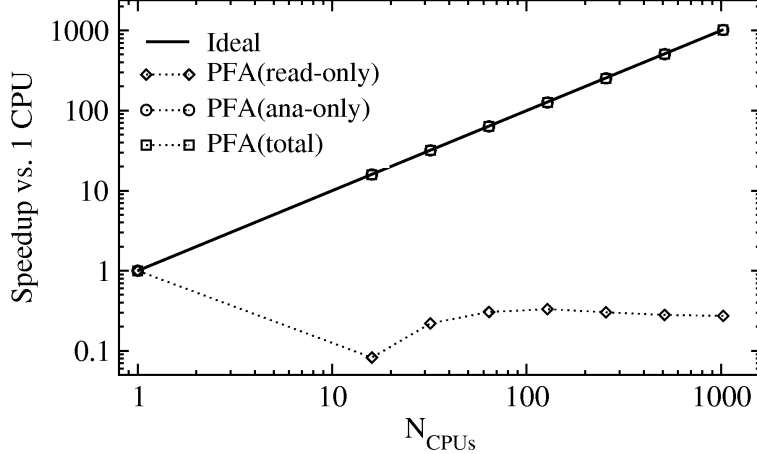


Fig. 3. Breakdown of the timing of PFA on the Ranger Supercomputer. These timing analysis have been measured using MPI’s WTIME function.

provided enough computations to illustrate the high efficiency of the algorithm at thousands of CPUs. The test analysis consisted of the difference dipole moment of the redox site and the center of mass distance between two electron transfer cofactors, as well as the Coulomb and Induction contributions to the vertical energy gap as given in equations 4–8. The results of the calculations have also been covered elsewhere[6], so only the scalability of the analysis itself will be discussed here. All timing data for the reading time, analysis time, and full software analysis time is provided in Table 1.

The speedup data for this test on Ranger is shown in Fig. 3. From the splitting of the performance data into its three principle components, one will notice that the speedup of the reading in parallel is quite poor (diamonds in Fig. 3). Even though the reading time generally decreased with increasing CPUs, the reading time cannot be decreased to less than around 60s for this particular test. However, the reading times are somewhat linearly decreasing in the range of  $N_{CPUs} = 16 - 128$  (see Table 1), suggesting that the shortness of the trajectory skews the reading parallelizability to appear less scalable. In other words, longer trajectories will lead to an increase in speedup because the time of reading the total trajectory is no longer on the order of the MPI IO initialization time, or more importantly, on the order of the analysis time.

In contrast to the poor scalability seen for reading, the analysis algorithm shows *embarrassingly parallel* scalability (circles in Fig. 3). Over the entire test range, from 1 – 1024 CPUs, the speedup of the analysis algorithm itself traces the ideal linear case. Ideal speedup is achieved for the analysis because of the large system size (ca. 50,000 atoms), the large number of pairwise calculations due to the invoked function calls, and the trajectory length is long enough to ensure reasonable performance on a large number of CPUs. Therefore, when analyzing a nanosecond or longer trajectory  $\tau_{read} \ll \tau_{ana}$ , and parallel reading no longer affects the scalability. However achieving this kind of speedup is

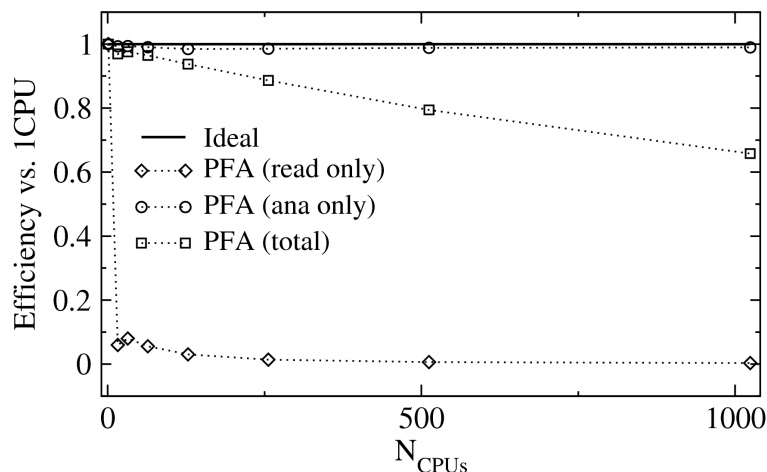


Fig. 4. Breakdown of the efficiency of PFA on the Ranger Supercomputer. Although the reading is considerably inefficient, the analysis algorithm itself is so efficient that the entire program is more than 98 % efficient over thousands of processors.

not unusual, but instead should be considered the minimum performance one can achieve with PFA. In fact, several calculations involving search routines over the first solvation shell of the protein are quite time consuming, and one can expect embarrassingly parallel scalability over larger number of processors with small systems and short ( $< 1$  ns) trajectories.

The poor scaling of the parallel reading degrades the overall performance of the total program (squares in Fig. 3) considerably. In the region of linear scaling of the reading times, generally less than a few hundred processors, the total program scales linearly as well. However, once the reading time cannot be lowered with a larger number of processors and it becomes roughly 1/3 of the total time of the total program, the program cannot scale due to the limitations of the IO interface.

As one would expect the parallel performance of the algorithm creates an environment for high efficiency, which can be defined as  $Speedup(N_{CPUs})/N_{CPUs}$  and is given for the Ranger test in Fig. 4. Despite the fact that the reading code slows to maximum inefficiency after a small number of CPUs (diamonds), the analysis algorithm is roughly 99 % efficient over 1024 CPUs. Such efficiencies are rarely reported, and point again to an algorithm with an embarrassingly parallel classification. On the other hand, the lack of scalability of the reading drags the overall performance down after a few hundred CPUs, and at the maximum CPU count is only about 65 % efficient.

The PFA algorithm has been shown to be highly optimized in terms of speedup and efficiency, yet one must wonder how it compares to other parallel software packages available for biological simulation analysis. To the author's knowledge, the only other parallel MD analysis program is the CHARMM molecular dynamics package. However, CHARMM is not optimized to run in a high

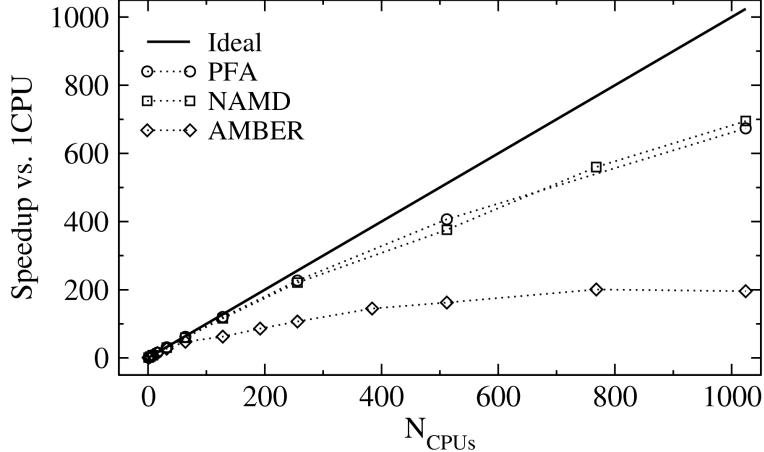


Fig. 5. Comparison of the parallel scalability of the analysis code against two leading MD software packages, NAMD and AMBER. The NAMD performance is measured for the Apo-1 protein, AMBER performance is for Cellulose, while PFA is tested using the BRC aqueous micelle.

performance environment of thousands of CPUs and therefore the comparison between PFA and CHARMM is not adequate. To date, two of the most scalable and canonical simulation packages, NAMD and AMBER, could be modified to incorporate the calculations already built into the PFA program. However, these packages employ a variety of computational tricks including smooth particle mesh Ewald[4] and multipole methods[1] for long range electrostatics, as well as spacial and particle-based parallel decompositions. This translates the system being essentially split amongst compute nodes, and in order to add analysis functionality, extra communication per simulation step would be necessary. This cost would inevitably lead to decreased performance, and one can imagine that keeping the analysis and simulation separate would lead to the most efficient use of computational resources. That said, a comparison will be made assuming the best possible scenario from AMBER and NAMD in order to compare the overall scalability of PFA.

Shown in Fig. 5 are the results for the total PFA program (circles), along with the best scalability results for both NAMD (squares) [13] and AMBER (diamonds)[14] on modern supercomputers. As can be seen, all three scale ideally to roughly 100 processors. However after about 300 CPUs all three programs lose scalability, with AMBER degrading the fastest with more CPUs. At the highest CPU count, AMBER is more than 3x less scalable than PFA, while PFA and NAMD are nearly equivalent. It should be noted that at 512 CPUs, PFA outperforms both NAMD and AMBER in terms of efficiency and speedup. Even more impressive, this comparison reports the performance metrics for AMBER and NAMD which are their respective best, while PFA used just a small trajectory with a rather small number of calculations per analysis run. This means that Gustafson's law can be invoked even further to provide increased scalability by increasing the calculations per analysis run,

or even more simply by increasing the trajectory length.

## 5 Conclusions

Overall, the PFA analysis algorithm has been shown to be embarrassingly parallel over thousands of CPUs on the Ranger Supercomputer. Due to the poor performance and efficiency of parallel reading, the total speedup of the PFA software degrades after a few hundred CPUs and is only about 65 % efficient at 1,024 compute nodes. In fact, the PFA software outperforms two MD software packages with similar analysis functionality at  $N_{CPU_s} > 300$ . The software, already involved in more than 10 CPU-years of production data analysis, will become more useful in time as more functions are added to the growing library, and as larger biological systems and long simulations times force the use of parallel algorithms into everyday life. One can envision the utility of this software in analysis of replica exchange simulations, simulations of virus capsids, as well as mixed biomolecular systems as found in membrane bound protein simulations.

## 6 Acknowledgments

I am forever indebted to my advisor, Dmitry Matyushov, whose years of enlightening discussions and financial support made this code possible. A special thanks goes to David Case and the Amber Development Team for granting permission of the use of their topology file reader code. Also, I would like to recognize Vitaliy Kapko who suggested the current data partitioning scheme, and for helping to fish out many mistakes with the energy gap calculations. D.N.L was supported by the NSF (CHE-0616646), and computer time at Ranger was provided by the TeraGrid's DAC program (TG-MCB080080N).

## References

- [1] Schlick, T. et al., Journal of Computational Physics **151** (1999) 9.
- [2] Case, D. A. et al., Journal of Computational Chemistry **26** (2005) 1668.
- [3] Brooks, B. et al., Journal of Computational Chemistry **4** (1983) 187.
- [4] Essmann, U. et al., The Journal of Chemical Physics **103** (1995) 8577.
- [5] LeBard, D. and Matyushov, D., Journal of Physical Chemistry B **112** (2008) 5218.

- [6] LeBard, D. N., Kapko, V., and Matyushov, D. V., Journal of Physical Chemistry B (2008).
- [7] Matyushov, D., Accounts of Chemical Research **40** (2007) 294.
- [8] LeBard, D. N. and Matyushov, D. V., The Journal of Chemical Physics **128** (2008) 155106.
- [9] Small, D., Matyushov, D., and Voth, G., Journal of the American Chemical Society **125** (2003) 7470.
- [10] Chaudhury, S. and Cherayil, B. J., The Journal of Chemical Physics **127** (2007) 145103.
- [11] Gustafson, J. L., Communications of the ACM **31** (1988) 532.
- [12] Jorgensen, W. L., Chandrasekhar, J., Madura, J. D., Impey, R. W., and Klein, M. L., The Journal of Chemical Physics **79** (1983) 926.
- [13] Brunner, R. K., Phillips, J. C., and Kale, L. V., Scalable molecular dynamics for large biomolecular systems, in *Proceedings of the IEEE/ACM SC2000 Conference*, page Technical Paper 271, IEEE Press, 2000.
- [14] Walker, R., A guide to running amber molecular dynamics simulations at san diego supercomputer center (sdsc),  
  
[http://coffee.sdsc.edu/rcw/amber\\_sdsc](http://coffee.sdsc.edu/rcw/amber_sdsc)

.